Avoidance algorithms

- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the banker's algorithm

Resource allocation algorithm (Banker's Algorithm)

- Multiple instances
- Each process must a priori claim maximum use
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return them in a finite amount of time

Data Structures for the Banker's Algorithm

- Let n = number of processes, and m = number of resources types.
- **Available**: Vector of length m. If available [j] = k, there are k instances of resource type R_j available
- **Max (*Resources*)**: $n \times m$ matrix. If $Max[i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation**: $n \times m$ matrix. If Allocation$[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need**: $n \times m$ matrix. If $Need[i,j] = k$, then P_i may need k more instances of R_j to complete its task

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

Example

Allocated

	R1	R2	R3
P1	1	1	0
P2	3	0	1
P3	3	1	2
P4	0	0	2

Claim

	R1	R2	R3
P1	5	2	2
P2	3	1	4
P3	5	1	3
P4	4	2	2

Resource

R1	R2	R3
9	3	6

Available

R1	R2	R3
2	1	1

Available

R1	R2	R3
0	1	0

Available

R1	R2	R3
5	2	3

Available

R1	R2	R3
5	1	0

allocated

	R1	R2	R3
P1	1	1	0
P2	3	0	1
P3	5	1	3
P4	0	0	2

allocated

	R1	R2	R3
P1	1	1	0
P2	3	0	1
P3	0	0	0
P4	0	0	2

allocated

	R1	R2	R3
P1	1	1	0
P2	3	1	4
P3	0	0	0
P4	0	0	2

Available

R1	R2	R3
8	2	4

allocated

	R1	R2	R3
P1	1	1	0
P2	0	0	0
P3	0	0	0
P4	0	0	2

Available

R1	R2	R3
4	0	4

allocated

	R1	R2	R3
P1	1	1	0
P2	0	0	0
P3	0	0	0
P4	4	2	2

Available

R1	R2	R3
8	2	6

Available

R1	R2	R3
4	1	4

allocated

	R1	R2	R3
P1	1	1	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

allocated

	R1	R2	R3
P1	5	2	2
P2	0	0	0
P3	0	0	0
P4	0	0	0

Available

R1	R2	R3
9	3	6

allocated

	R1	R2	R3
P1	0	0	0
P2	0	0	0
P3	0	0	0
P4	0	0	0

- It is safe state, because all processes can complete its work and finish.

Example: is the following case in a safe state

allocated المخصص

	R1	R2	R3
P1	2	0	1
P2	5	1	1
P3	2	1	1
P4	0	0	2

Claim المطلوب

	R1	R2	R3
P1	3	2	2
P2	6	1	3
P3	3	1	4
P4	4	2	2

Available المتاح

R1	R2	R3
o	1	1

- P1 = 1+2+1 = 4
- P2 = 1+2 = 3
- P3 = 1+3 =4
- P4 = 4+2 = 6

It is not safe state because all processes can not complete its work and finiṡh

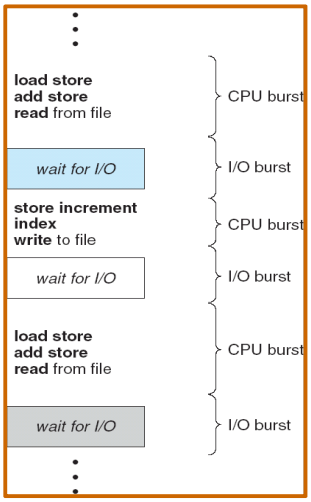
Deadlock Recovery

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated. In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?
- Resource preemption
  - Selecting a victim – minimize cost.
  - Rollback – return to some safe state, restart process for that state.
  - Starvation – same process may always be picked as victim - include number of rollback in cost factor.

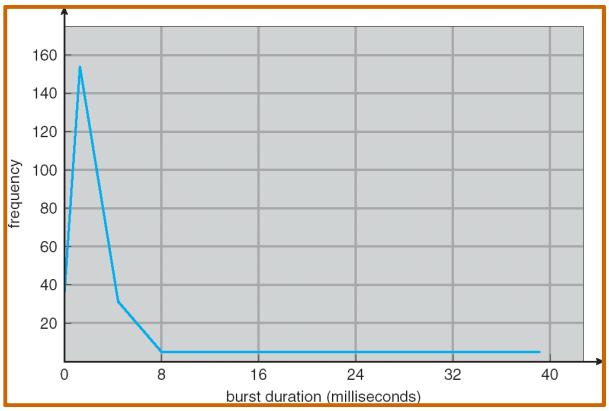
٥٦

CPU Scheduling

- Maximum CPU utilization is obtained with multiprogramming
  - Several processes are kept in memory at one time
  - Every time a running process has to wait, another process can take over use of the CPU
- Scheduling of the CPU is fundamental to operating system design
- Process execution consists of a *cycle* of a **CPU time burst** and an **I/O time burst** (i.e. wait) as shown on the next slide
  - Processes alternate between these two states (i.e., CPU burst and I/O burst)
  - Eventually, the final CPU burst ends with a system request to terminate execution



Histogram of CPU-burst Times



CPU bursts tend to have a frequency curve similar to the exponential curve shown above. It is characterized by a large number of short CPU bursts and a small number of long CPU bursts. An I/O-bound program typically has many short CPU bursts; a CPU-bound program might have a few long CPU bursts.

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is **nonpreemptive;** they offer no schedule choice.
- All other scheduling is **preemptive;** they can be scheduled

٥٩

Dispatcher

- The dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- The dispatcher needs to run as fast as possible, since it is invoked during process context switch
- The time it takes for the dispatcher to stop one process and start another process is called **dispatch latency**

٦٠

Scheduling Criteria

- Different CPU scheduling algorithms have different properties
- Criteria for comparing CPU scheduling algorithms may include the following
  - **CPU utilization** – percent of time that the CPU is busy executing a process (Maximize CPU utilization)
  - **Throughput** – number of processes that are completed per time unit (Maximize throughput))
  - **Response time** – amount of time it takes from when a request was submitted until the first response occurs (Minimize response time)
  - **Waiting time** –the sum of the amount of time a process has spent waiting in the ready queue. (Minimize waiting time)
  - **Turnaround time** – amount of time to execute a particular process from the time of submission through the time of completion (Minimize turnaround time)

٦١

Single Processor Scheduling Algorithms

- First Come, First Served (FCFS)
- Shortest Job First (SJF)
- Priority
- Round Robin (RR)

First Come, First Served (FCFS)

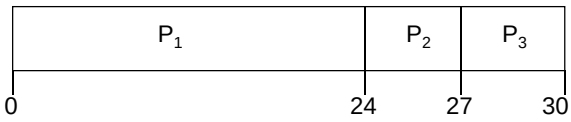
- With FCFS, the process that requests the CPU first is allocated the CPU first
- The FCFS scheduling algorithm is **non-preemptive**
- Once the CPU has been allocated to a process, that process keeps the CPU until it releases it either by terminating or by requesting I/O

٦٢

First Come, First Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



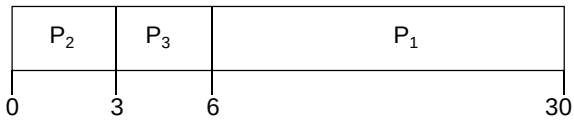
- Waiting time for P1 = 0; P2 = 24; P3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

٦٣

Suppose that the processes arrive in the order

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6; P_2 = 0, P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case
- *Convoy effect* short process behind long process

٦٤

Shortest-Job-First (SJF) Scheduling

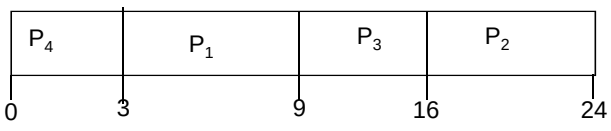
- The SJF algorithm associates with each process the length of its next CPU burst
- When the CPU becomes available, it is assigned to the process that has the smallest next CPU burst (in the case of matching bursts, FCFS is used)
- Two schemes:
  - **Nonpreemptive** – once the CPU is given to the process, it cannot be preempted until it completes its CPU burst
  - **Preemptive** – if a new process arrives with a CPU burst length less than the remaining time of the current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request

١٠

Example of SJF (non-preemptive - simultaneous arrival (arrival-time=0))

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



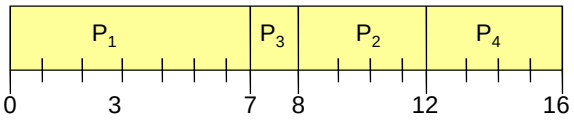
- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

١١

Example : non-preemptive SJF (varied arrival times)

Process	Arrival Time	Burst Time (assume millisecond)
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive, varied arrival times)



- Average waiting time = ((0 – 0) + (8 – 2) + (7 – 4) + (12 – 5))/4
  = (0 + 6 + 3 + 7)/4 = 4 ms
- Average turn-around time: = ((7 – 0) + (12 – 2) + (8 - 4) + (16 – 5))/4
  = (7 + 10 + 4 + 11)/4 = 8 ms

٦٧

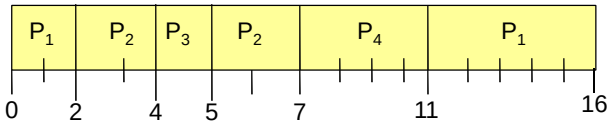
Preemptive SJF [Shortest-remaining-time-first] (SRTF).

- Yank the CPU away from the currently executing process when a higher priority process is ready.
- Avoids "hogging" of the CPU
- On time sharing machines, this type of scheme is required because the CPU must be protected from a run-away low priority process.
- Give short jobs a higher priority – perceived response time is thus better.

٦٨

Example: Preemptive SJF [Shortest-remaining-time-first] (SRTF - varied arrival times)

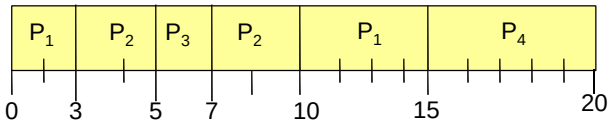
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4



- Average waiting time
  = ([(0 – 0) + (11 - 2)] + [(2 – 2) + (5 – 4)] + (4 - 4) + (7 – 5))/4
  = 9 + 1 + 0 + 2)/4 =3 ms
- Average turn-around time = (16 + 5 + 1 + 6)/4 = 7 ms

Example: Preemptive SJF [Shortest-remaining-time-first] (SRTF - varied arrival times)

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	3.0	5
P_3	5.0	2
P_4	6 .0	5



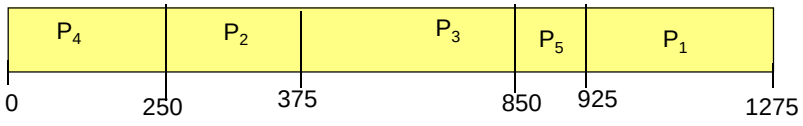
- Average waiting time
  = ([(0 – 0) + (10 - 3)] + [(3 – 3) + (7 – 5)] + (5 - 5) + (15 – 6))/4
  = 7+ 0 + 2 +0+ 9)/4 =4.5ms
- Average turn-around time = (15+ 7 + 2 + 14)/4 =9.5 ms

Priority Scheduling

- The SJF algorithm is a special case of the general priority scheduling algorithm
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
- Priority scheduling can be either preemptive or non-preemptive
  - A **preemptive** approach will preempt the CPU if the priority of the newly-arrived process is higher than the priority of the currently running process
  - A **non-preemptive** approach will simply put the new process (with the highest priority) at the head of the ready queue
- SJF is a priority scheduling algorithm where priority is the predicted next CPU burst time
- The main problem with priority scheduling is **starvation**, that is, low priority processes may never execute
- A solution is **aging**; as time progresses, the priority of a process in the ready queue is increased

Example (non-preemptive)

Process	Burst Time	Priority
P1	350	5
P2	125	2
P3	475	3
P4	250	1
P5	75	4



- Average waiting time = (0 + 250 + 375 + 850+925) / 5 = 480 ms

Round Robin (RR) Scheduling

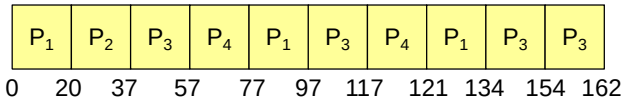
- In the round robin algorithm, each process gets a small unit of CPU time (a *time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance of the round robin algorithm
  - q large $\Rightarrow$ FCFS
  - q small $\Rightarrow$ q must be greater than the context switch time; otherwise, the overhead is too high

٧٣

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

The Gantt chart is:



Typically, higher average turnaround than SJF, but better response time

- Average waiting time
  = ([(0 – 0) + (77 - 20) + (121 – 97)] + (20 – 0) + [(37 – 0) + (97 - 57) + (134 – 117)] + [(57 – 0) + (117 – 77)]) / 4
  = (0 + 57 + 24) + 20 + (37 + 40 + 17) + (57 + 40)) / 4
  = (81 + 20 + 94 + 97)/4
  = 292 / 4 = 73 ms

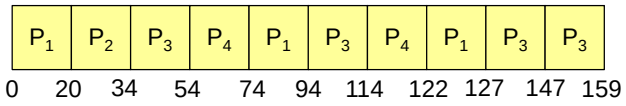
٧٤

- Average turn-around time = 134 + 37 + 162 + 121) / 4 = 113.5 ms

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	45
P_2	14
P_3	72
P_4	28

The Gantt chart is:



Typically, higher average turnaround than SJF, but better response time

- Average waiting time
  = ([(0 – 0) + (74 - 20) + (122 – 94)] + (20 – 0) + [(34 – 0) + (94 - 54) + (127 – 114)] + [(54 – 0) + (114 – 74)]) / 4
  = (0 + 54 + 28) + 20 + (34 + 40 + 13) + (54 + 40)) / 4
  = (82 + 20 + 87 + 94)/4
  = 283 / 4 = 70.3 ms
- Average turn-around time = 127+ 34 + 159 + 122) / 4 = 478/4 =119.5 ms